

## Parallelizing AT with open multi-processing and MPI\*

LUO Cheng-Ming (罗承明),<sup>1,2</sup> TIAN Shun-Qiang (田顺强),<sup>1</sup> WANG Kun (王坤),<sup>1</sup>  
ZHANG Man-Zhou (张满洲),<sup>1</sup> ZHANG Qing-Lei (张庆磊),<sup>1,2</sup> and JIANG Bo-Cheng (姜伯承)<sup>1,†</sup>

<sup>1</sup>Shanghai Institute of Applied Physics, Chinese Academy of Sciences, Shanghai 201800, China

<sup>2</sup>University of Chinese Academy of Sciences, Beijing 100049, China

(Received September 26, 2014; accepted in revised form March 5, 2015; published online June 20, 2015)

Simulating charged particle motion through the elements is necessary to understand modern particle accelerators. The particle numbers and the circling turns in a synchrotron are huge, and a simulation can be time-consuming. Open multi-processing (OpenMP) is a convenient method to speed up the computing of multi-cores for computers based on share memory model. Using message passing interface (MPI) which is based on non-uniform memory access architecture, a coarse grain parallel algorithm is set up for the Accelerator Toolbox (AT) for dynamic tracking processes. The computing speedup of the tracking process is 3.77 times with a quad-core CPU computer and the speed almost grows linearly with the number of CPU.

Keywords: Accelerator Toolbox, Open multi-processing, Message passing interface, Parallel computing

DOI: 10.13538/j.1001-8042/nst.26.030104

## I. INTRODUCTION

The Accelerator Toolbox (AT) code was developed at the Stanford Synchrotron Radiation Lightsource (SSRL) to model particle accelerators and beam transport lines in the MATLAB environment [1]. Users of AT can develop their own functions and applications to meet their various requirements by building on the AT source code. The AT results agree well with experimental measurements. At Shanghai Synchrotron Radiation Facility (SSRF), AT has been used for several years [2, 3]. In storage ring simulations, dynamic aperture tracking and lattice design optimization require highly computation-intensive algorithms. To finish the computation in an acceptable time, one needs to make the code more efficient. The bottlenecks in AT performance come from massive repeated calls of particle tracking functions which are independent of each other. So paralleling computing is a straightforward way to hasten the computing speed [4].

There are two common ways to parallel a program: using graphics processing unit (GPU) to compute, and using multi-core CPU to start multi-thread computation. Since the computing processes of AT's tracking program involve lots of cache operations, the frequent exchange data between the computer memory and GPU makes GPU computing less attractive when the number of particles is relatively small. So using multi-core CPU to compute is a better option. Open multi-processing (OpenMP) is a standardized model to share memory computing that supports C/C++ compiler. On a local quad-core computer, one is able to achieve a speed increase of 3.7 times. Based on message passing model, message passing interface (MPI) is used on a dual CPU server to further increase the speed. And the speed of computing grows linearly with the number of computers.

## II. EXPERIMENTAL

## A. Accelerator physics and optimization analysis

The particle motion is modeled by using a point in 6-dimensional phase space coordinates to represent a particle in AT.

$$\mathbf{X} = (x, p_x, y, p_y, (p - p_0)/p_0, c\tau)^T, \quad (1)$$

where,  $x$  and  $y$  are the transverse coordinates,  $p_x$  and  $p_y$  are the divergences of  $x$  and  $y$ ,  $(p - p_0)/p_0$  is the momentum spread, and  $c\tau$  is the longitudinal position. Evolution of the phase space point through a magnetic accelerator-element can be modelled using the second order transport matrix [5]

$$\mathbf{X}_j^{\text{final}} = \Delta \mathbf{X}_j + \sum_{k=1}^6 \mathbf{R}_{jk} \mathbf{X}_k + \sum_{k=1}^6 \sum_{l=1}^6 \mathbf{T}_{jkl} \mathbf{X}_k \mathbf{X}_l. \quad (2)$$

Therefore, many multiply-reduced operations can be required to compute the entire process evolving a particle through a single magnetic element. The particles are assumed to be sufficiently relativistic that inter-particle interactions can be ignored, allowing the same operation to be applied to all particles in parallel [5]. Different accelerator element has different transport matrix. In AT code, they are calculated in different 'passmethod' functions, which are called million times in tracking process by AT's 'ringpass' function to calculate the particle trajectories. Calculating multi-particle through the accelerator element is a SIMD (single instruction multiple data) operation, and thus well suited to OpenMP and MPI processing [6].

Parallel computing uses multiple computing resources to work simultaneously on different parts of a problem [7]. It is an efficient way to speed up computing. We focus mainly on making AT compatible with parallel processing with OpenMP and MPI. OpenMP is an application programming interface (API) for multi-thread programming in C/C++ and FORTRAN. It offers a highly abstract description of parallel

\* Supported by the National Natural Science Foundation of China (No. 11105214)

† Corresponding author, jiangbocheng@sinap.ac.cn

computing. It is composed of a set of compiler directives, library routines and environment variables affecting run-time behavior [8]. By introducing OpenMP routines and directives to the existing AT source code, we make AT follow a uniform memory access (UMA) model, in which all the cores of processors share the same physical memory uniformly. It requires moderate changes to the 'passmethod' functions written in C language, for which OpenMP can be implemented by Matlab's MEX compiling function [9].

MPI is the standard of distribution model using explicit ways to control parallel computing. MatlabMPI is a Matlab implementation of the MPI standard and allows any Matlab program to exploit multiple processors [10]. A non-uniform memory access (NUMA) architecture is built with OpenMP and MPI. In this model, multi machines run independently with its own local memory and communicate between each other with Bus Interconnect. Fig. 1 shows schematically the NUMA model.

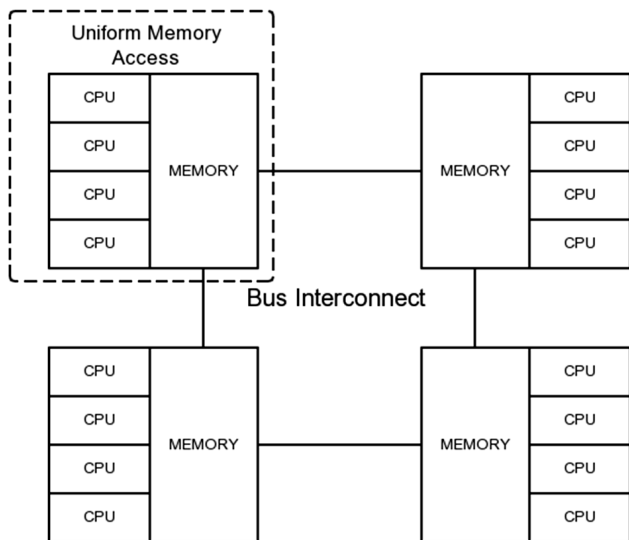


Fig. 1. Schematics of non-uniform memory access (NUMA).

In this case, we use a general distributed memory model. The upper computers use MatlabMPI to control the parallel computing and communication, and the lower computers use OpenMP to compute. With message-passing functions, the data are distributed to lower computers, and the computing results from the lower computers are then transferred to the upper computers and combined into the final result. This pattern allows us to use the shared memory computing interface to manage local task distribution for every CPU, and makes AT functioning in a global distributed memory model. In this way, we can avoid the data conflict caused by two processors trying to access to the same memory. If only UMA model is used, such a conflict will cause a longer CPU spin time. In some cases, the data-conflict delay can make working time of two processors longer than that of one processor.

## B. Parallel AT with OpenMP and MPI

As mentioned above, the speed increase of *passmethod* functions will increase speed of computing. OpenMP and MPI are used to parallelize the *passmethod* functions. The way to create an OpenMP and MPI program with existing code is to find sections of the codes that can be processed simultaneously. The changes in the codes and the way to compile the codes are given in the appendix. The parallel part of the function use library functions *omp\_get\_num\_threads()* and *omp\_get\_thread\_num()* to get the number of threads in the parallel zone and the id of the working thread (all the threads are numbered from zero). The *start\_index* is the offset for each computing core. According to the thread id number, different thread works on different data. In this way, parallelizing the data and tasks are realized.

Taking DriftPass.c [2], which is used to calculate the status after the particles passing through a drift element, as an example. Intel VTune Amplifier [11] is used to test efficiency of the parallel DriftPass.c. The number of particles is 3920, with 500000 loops on a qual-core computer. The results are given in Table 1, where the CPU time is the sum of CPU time of all threads, and the overhead & spin time is the time an active thread takes to get a synchronous construct. These two make up most of the CPU time. The overhead & spin time takes about 9.3% of all CPU time. That is the main reason the speedup of a parallel program cannot reach the limit value of 4. The speedup is 2.98.

TABLE 1. Computation time for parallel and non-parallel DriftPass

Type of computing	Elapsed time (s)	CPU time (s)	Overhead & spin time (s)
Parallel	10.28	77.83	7.22
Non-parallel	30.09	30.05	0.00

OpenMP is used to parallelize all the 'passmethod' functions called in 'ringpass', with all of the 'passmethod' codes being parallelizable. Therefore *ringpass* can be treated as a parallel program.

## III. RESULTS AND DISCUSSION

Frequency map analysis (FMA) is an analysis method to find the amplitude of frequency shifts within a dynamic aperture. The program flow traces the particles, obtains output data of the particles through  $N$  turns, and uses a first order Hamming filter to filter the data [12]. FMA is applied as a frequency scanning tool to reveal information about nonlinear resonances and guide frequency optimization [13]. The particle tracking takes most of the computing time. OpenMP is used to reprogram AT to shorten the time, which may save days or weeks. Figure 2 shows the result of using parallel and non-parallel methods to compute FMA with different numbers of particles using an Intel i7-3770 CPU with 4G RAM.

The FMA execution time grows almost linearly with the number of particles. The non-parallel method is up to 3.16

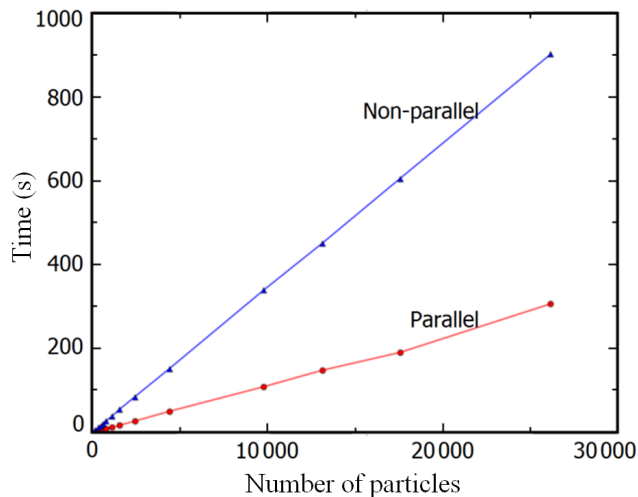


Fig. 2. FMA execution time for non-parallel and parallel computing as function of number of particles.

TABLE 2. Time profile (in second) using parallel and non-parallel computing

$N$	Parallel computing			Non-parallel computing		
	$T_f$	$T_{all}$	$T_f/T_{all}$	$T_f$	$T_{all}$	$T_f/T_{all}$
512	1.19	5.87	0.202	1.25	17.52	0.071
1128	2.66	12.58	0.207	2.82	39.11	0.072
2450	5.75	28.04	0.205	6.16	85.92	0.071
4418	10.18	51.28	0.206	11.01	152.52	0.072
9800	22.47	109.74	0.207	24.40	339.43	0.072
177578	50.57	191.24	0.212	43.50	604.51	0.072

times slower than the parallel method. According to Amdahl's law [14]

$$S = [f_{\text{par}}/P + (1 - f_{\text{par}})]^{-1}, \quad (3)$$

where  $f_{\text{par}}$  is the parallel fraction of the code,  $P$  is the speedup for the parallel part, and  $S$  is the speedup of the whole program. The profile command is used to obtain the time for parallel and non-parallel parts of the program flow. The parallel part is the 'ringpass' function and the main non-parallel part is the FMA function. The remaining parts take little of the total time. Table 2 shows the results for the parallel and non-parallel programs, with  $N$  being the number of particles,  $T_f$  the time the FMA function takes and  $T_{\text{all}}$  the time for the whole program.

From Table 2, the  $T_f/T_{\text{all}}$  ratio is stable for both types of computing. According to Amdahl's law, the speedup of the parallel part can be obtained by Eq. (4)

$$[(1 - 0.07)/P + 0.07]^{-1} = 3.16, \quad (4)$$

where 0.07 is the non-parallel part of the computing process. So, the speedup is  $P = 3.77$ , and it never exceeds 4, with even larger number of particles. The CPU is quad-core, so there will be a maximum of 4 computing threads executing at any one time, and synchronization between the threads also reduces the compute speed.

To take advantage of the speedup factor increase by OpenMP and MPI, a Dell R720 server is utilized for particle tracking in the slow extraction of Shanghai Proton Therapy Synchrotron. R720 has 2 processors which has 16 CPU cores each. It acts as two compute nodes in the computing process. The speedup of one node is 6.23. The speedup of 2 nodes is 12.18 which is almost double of the speedup of one node. Since the computing process is independent of each other, and the communication takes about 2.3% of the total running time, the speedup of two nodes should almost double the speedup of one node. It can be estimated that the speedup of the computing process can grow linearly with the number of CPU using OpenMP and MPI.

#### IV. CONCLUSION

In this paper, we have introduced the way AT works, how OpenMP and MPI can be used in parallelizing programs. The parallelized AT can compute faster. If the code can be parallelized, OpenMP and MPI can be used in a similar way for other accelerator physics programs. This pattern is convenient to use and the speedup is close to the limit of what can be achieved by a single computer or a cluster. With more computer nodes, larger problems can be solved.

#### APPENDIX

```
#include<omp.h>
..... some computation and initialization
Omp_set_num_threads(4)
#pragma omp parallel private (i) share ( start_index ,n)
{
    thread_id=omp_get_thread_num();
    num_threads=omp_get_num_threads();
    start = start_index + n*thread_id /num_threads;
    if ( thread_id==num_threads-1)
    end=n-1;
    else
    end=n*(thread_num+1)/ num_threads-1;
    for (i= start ; i<=end;i++){
        ... computation
    }
}
```

[1] Terebilo A. Accelerator toolbox for MATLAB. SLAC-PUB-8732, 2001.

[2] Tian S Q, Jiang B C, Zhou Q G. Lattice design and optimization of the SSRF storage ring with super-bends. Nucl Sci Tech,

- 2014 **25**: 010102. DOI: [10.13538/j.1001-8042/nst.25.010102](https://doi.org/10.13538/j.1001-8042/nst.25.010102)
- [3] Jiang B C, Liu G M, Zhao Z T. Simulation of a transverse feedback system for the SSRF storage ring. *High Energ Phys Nucl*, 2007, **31**: 956–961.
- [4] Yan X Y, Zhang W W, Bu S H. Parallel optimization of three-dimension particle simulation based on mixed MPI/OpenMP Programming. *Journal of South China University of Technology (Natural Science Edition)*, 2012, **40**: 71–78. DOI: [10.3969/j.issn.1000-565X.2012.04.011](https://doi.org/10.3969/j.issn.1000-565X.2012.04.011)
- [5] Grote H, Iselin F C. The MAD Program (Methodical Accelerator Design) Version 8.13/8 User's Reference Manual. Geneva, Switzerland. Jan. 18, 1994.
- [6] Appleby R, Bailey D, Higham J, *et al.* High performance stream computing for particle beam transport simulations. *J Phys Conf Ser*, 2008, **119**: 042001. DOI: [10.1088/1742-6596/119/4/042001](https://doi.org/10.1088/1742-6596/119/4/042001)
- [7] Chen G L, Sun G Z, Xu Y, *et al.* Integrated research of parallel computing: Status and future. *Chinese Sci Bull*, 2009, **54**: 1845–1853. DOI: [10.1007/s11434-009-0261-9](https://doi.org/10.1007/s11434-009-0261-9)
- [8] Dagum L and Menon R. OpenMP: an industry standard API for shared-memory programming. *IEEE Comput Sci Eng*, 1998, **5**: 46–55. DOI: [10.1109/99.660313](https://doi.org/10.1109/99.660313)
- [9] Zhang Y. Solving large-scale linear programs by interior-point methods under the Matlab Environment. *Optim Method Softw*, 1998, **10**: 1–31. DOI: [10.1080/10556789808805699](https://doi.org/10.1080/10556789808805699)
- [10] Kepner J. Parallel programming with MatlabMPI. [arXiv: astro-ph/0107406](https://arxiv.org/abs/astro-ph/0107406)
- [11] Marowka A. On performance analysis of a multithreaded application parallelized by different programming models using intel Vtune. *Lect Notes Comput Sc*, 2011, **6873**: 317–331. DOI: [10.1007/978-3-642-23178-0\\_28](https://doi.org/10.1007/978-3-642-23178-0_28)
- [12] Laskar J. Frequency map analysis and particle accelerators. *IEEE Part Acc Conf*, 2003, **1**: 378–382. DOI: [10.1109/PAC.2003.1288929](https://doi.org/10.1109/PAC.2003.1288929)
- [13] Tian S Q, Liu G M, Li H H, *et al.* Tune optimization of the third generation light source storage ring based on Frequency Map Analysis. *Chinese Phys C*, 2009, **33**: 224–. DOI: [10.1088/1674-1137/33/3/012](https://doi.org/10.1088/1674-1137/33/3/012)
- [14] Chandra R, Dagum L, Kohr D, *et al.* Parallel programming in OpenMP. San Francisco (USA): Morgan Kaufmann Publishers, 2001, 16–17.